Audio Units and Audio Codecs

Session 508

Audio Units and Audio Codecs

Craig Keithley
USB and FireWire Technology Evangelist

Audio Codecs

Jeff Moore
Core Audio Engineering

Introduction

- Audio Codecs provide a plug-in mechanism for encoding and decoding audio data
- Audio Codecs are Components
- In Jaguar, the Audio Converter will use Audio Codecs as a user-extensible mechanism

Kinds of Audio Codecs

- There are three kinds of Audio Codecs
  - Encoders (‘aenc’) transform linear PCM data into another format
  - Decoders (‘adec’) transform other formats into linear PCM
  - Unity codecs (‘acdc’) transform between variants of the same format (e.g., a sample rate converter)

Audio Codec Discovery

- Audio Codecs are discovered like other components using Component Manager routines

  FindNextComponent

- Component, once found, is opened

  OpenAComponent

- The Component is closed when you are done

  CloseComponent

Audio Codec Properties

- Properties provide a means to configure the transformation performed by an Audio Codec
- The value of a property is an untyped block of memory whose contents are specified by the property’s ID

Audio Codec Properties

- Important properties:
  - **kAudioCodecPropertyRequiresPacketDescription**
  - **kAudioCodecPropertyPacketFrameSize**
  - **kAudioCodecPropertyHasVariablePacketByteSizes**
  - **kAudioCodecPropertyMaximumPacketByteSize**

Audio Codec Properties

- Important properties:
  - kAudioCodecPropertyCurrentInputFormat
  - kAudioCodecPropertyCurrentOutputFormat
  - kAudioCodecPropertyMagicCookie

Audio Codec States

- Audio Codecs can be in two states, uninitialized and initialized
- **AudioCodecInitialize** and **AudioCodecUninitialize** move between the states
- The state transition is when the codec allocates/ releases any resources it needs like buffers and tables

Audio Codec States

- When the codec is initialized, the parameters of the transformation cannot be changed
- Properties that depend on the configuration of the codec, like the maximum packet byte size, are only valid when the codec is initialized

Audio Codec Data Flow

- Audio Codecs uses a “push then pull” model for data flow
- Input data is provided using **AudioCodecAppendInputData**
  - Input data is copied into an internal buffer
  - The codec will return how much data it consumes
  - The data must be in full packets if packet descriptions are required

Audio Codec Data Flow

- Output data is produced using **AudioCodecProduceOutputPackets**
  - Output is always in full packets of data
  - A status code is returned indicating how much of the request could be satisfied

AudioCodec SDK

- A C++ class library for implementing Audio Codec components

Demo

Audio Codec SDK

Audio Units

Doug Wyatt
Core Audio Engineering

Introduction: Audio Units

- Functionality and packaging
- Writing an Audio Unit
- Writing an Audio Unit View

Audio Unit Functionality

- Audio signal-processing plug-in
- Optional user interface (view) plug-in
- Any number of input and output connections (busses)
- Pull model allows complex graphs (see AUGraph)
- Operates on 32-bit floating point buffers

Audio Unit Packaging

- Component bundle
  - Can contain resources
- Discover

  FindNextComponent
- Open

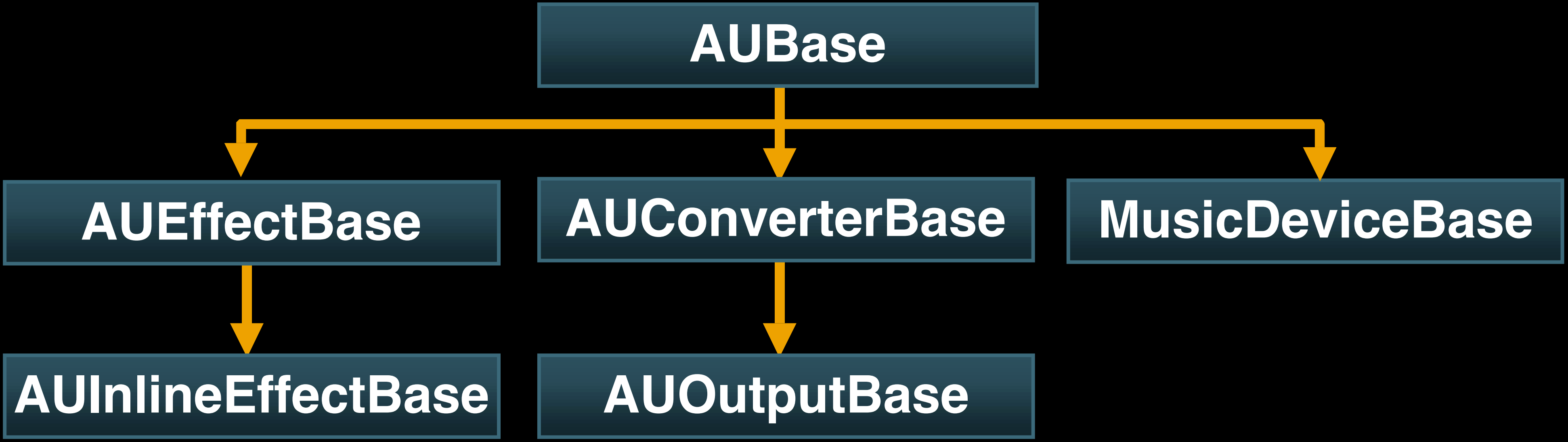
  OpenAComponent
- Close

  CloseComponent

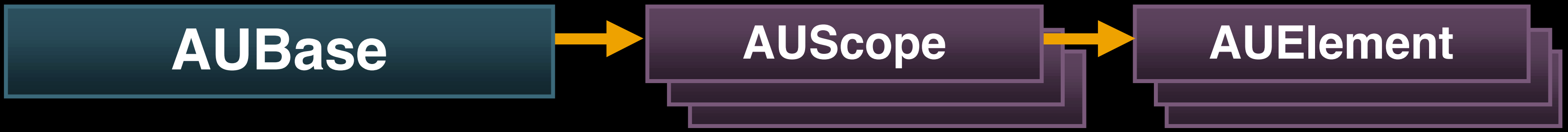
First Steps

- Determine numbers of input and output busses
- Each bus can be mono or multi-channel
    - 10.1 (current): Interleaved
    - Version 2, multi-channel busses are deinterleaved
- Choose an appropriate C++ base class from the SDK

SDK Class Hierarchy

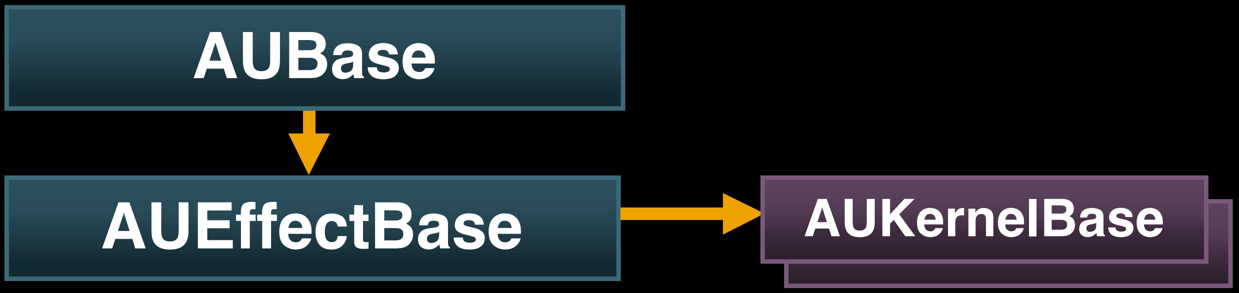


AUBase



- Translates component entry selectors and get/set property calls into C++ virtual methods
- Manages scopes (input, output, global) and elements
- Handles a significant amount of other housekeeping

AUEffectBase

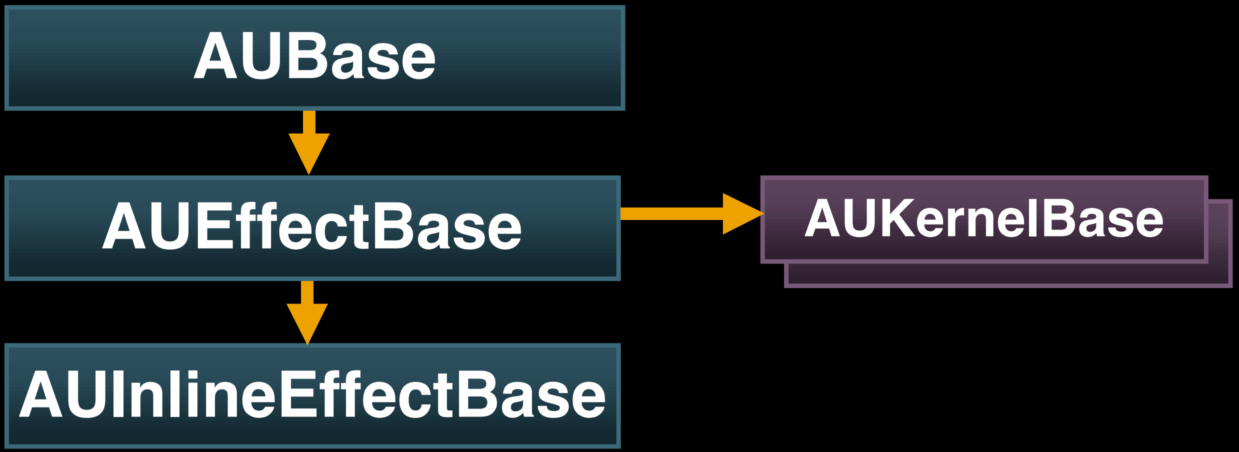

AUBase

AUEffectBase

AUKernelBase

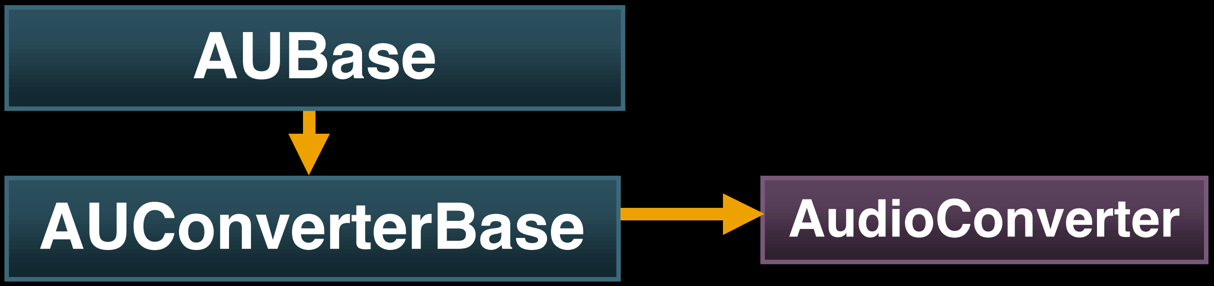
- 1 bus (element) in, 1 bus out, each with same number of channels
- Creates a “kernel” object per channel
- Default Render() implementation calls kernels to process in mono for each channel

AUInlineEffectBase



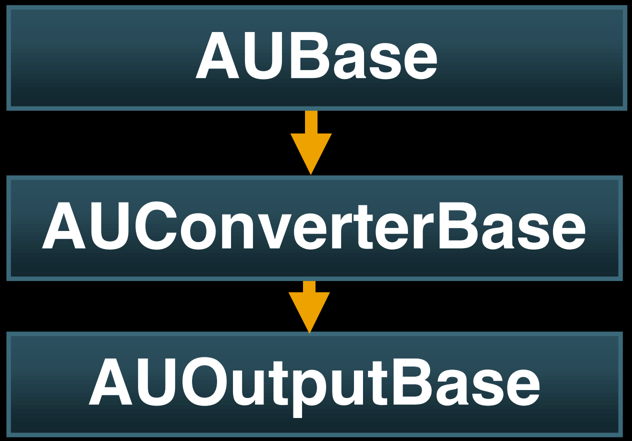
- Use this if your DSP can operate on samples in place
    - Provides cache efficiency

AUConverterBase



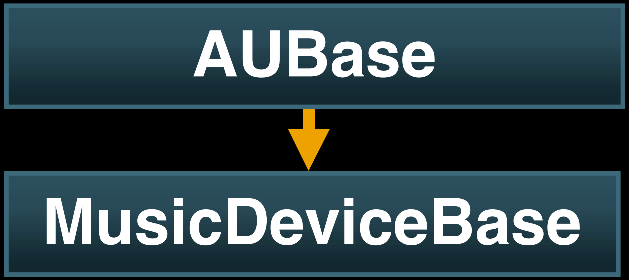
- Typical uses
  - Sample rate conversion
  - Interleaving/deinterleaving
  - Channel mapping

AUOutputBase



- Used for terminal nodes in graphs
    - E.g., hardware output units, file writers
- Supports Start and Stop methods
- Provides format conversion between a canonical float format and a hardware or file format

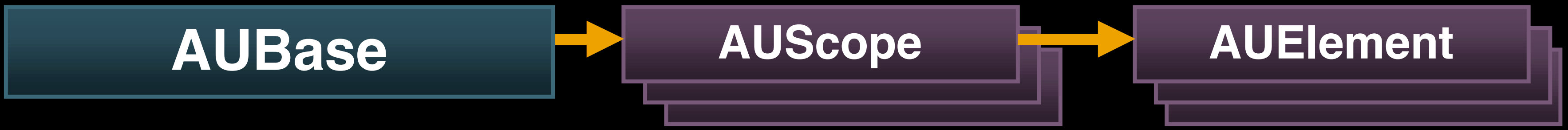
MusicDeviceBase


AUBase

MusicDeviceBase

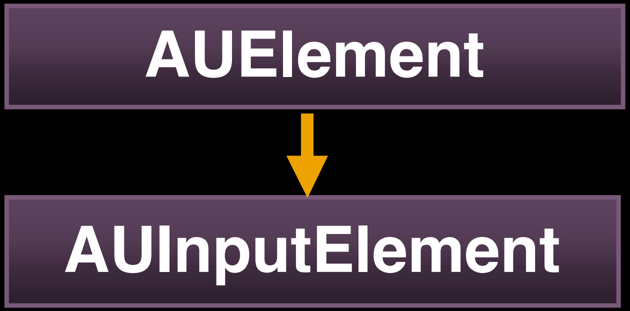
- Implements Music Device selectors
  - MIDI events
  - Extended note and control events
- Renders audio output like any other AudioUnit
- Use for soft synths

AUElement



- Audio Unit API refers to elements within scopes
- Managed by AUBase, but you can subclass them to store state per input/output
- Manages parameters and stream formats

AUInputElement

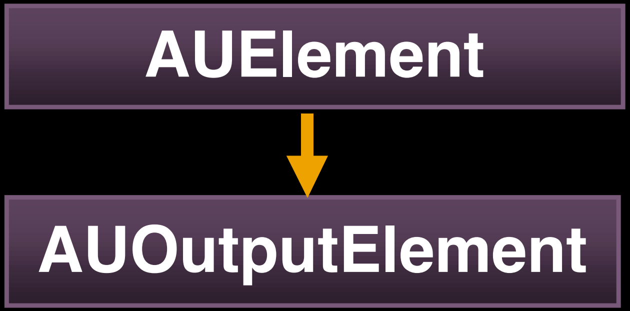


AUElement

AUInputElement

- Obtains input from upstream Audio Unit or a client callback function

AUOutputElement


AUElement

AUOutputElement

- Maintains buffers for caching output (to support fan-out connections)

Demo

Writing an Effect

AudioUnit V2

- Component description changes
- Different data structure passed in rendering process

Component Description Differences

- V1 (10.1)
    - ‘aunt’ component type, and multiple required subtypes
- V2
    - Multiple component types:
        - ‘aufx’, ‘auou’, ‘aumd’
    - Leaves manufacturer and sub-type available for your use
        - May have recommendations for sub-type usage

Differences in Rendering

- V1
  - Interleaved buffers for multichannel streams, in AudioBuffer structure
- V2
  - Deinterleaved buffers, in **AudioBufferList**
    - **AudioBufferList** contains N **AudioBuffers**
- Affects:
  - **RenderSlice** becomes **Render**
  - Render notification and input procs change accordingly

Deinterleaved–Why?

- You asked, we listened
- Much existing DSP code uses deinterleaved buffers
- Improved cache efficiency for many algorithms
- Simpler to optimize for multichannel and complex signal chain assembly

Implications of Changes—AUs

- Do not feel compelled to support V1 API
    - New Apple units after Jaguar will only be published with new component types
- To support both V1 and V2 in your AudioUnit, use component aliases to have two component descriptions
    - AUEffectBase and AUConverterBase provide support for both APIs
    - Other AUBase subclasses need to do more work to support both

Implications of Changes–Clients

- Can not mix V1 and V2 types in a graph
  - Would get stream format mismatches
- Client API changes to use **AudioBufferList** instead of **AudioBuffer**
  - **AudioUnitRender** replaces **AudioUnitRenderSlice**
  - Render and input callbacks have different signatures

AudioUnitCarbonView

- UI for an Audio Unit
- Also a component; Apple supplies generic view
- Audio Unit can specify, via a property, one or more view components that know how to control it
- Creates a Carbon user pane
  - Can contain controls or a custom UI
  - Uses Carbon Events
- Here too we supply a small C++ framework

AudioUnitCocoaView?

- We are working on it, but post-Jaguar
- Tricky issues of invoking Carbon views from Cocoa and vice versa
    - Want to make this transparent to the app

AUCarbonViewBase

- Handles the single-selector component interface:

```
AudioUnitCarbonViewCreate(
AudioUnitCarbonView inView,
AudioUnit                 inAudioUnit,
WindowRef                 inWindow,
ControlRef                inParentControl,
const Float32Point *      inLocation,
const Float32Point *      inSize,
ControlRef *              outControl);
```

- Manages binding of controls to parameters

Parameter Listeners

- Mechanism for receiving notifications when Audio Unit parameter values change
    - **AUParameterListener**
- Call **AUParameterSet** instead of **AudioUnitSetParameter**
- One listener can listen to multiple parameters
- See **AudioToolbox/AudioUnitUtilities.h**

Demo

Writing An AudioUnitCarbonView

Conclusion

- SDK base classes will support both versions of the API
    - So no reason not to start writing AudioUnits now
- Thanks for your feedback; please continue

Roadmap

502 Core Audio Technologies	Room J **Tue., 2:00pm**
507 Audio and MIDI: Using the Audio HAL and Core MIDI Services	Room J **Wed., 3:30pm**
FF014 Audio and MIDI: Question and Answer Forum	Room J **Fri., 3:30pm**
607 QuickTime and MPEG-4: Short Overview of AAC	Room A2 **Fri., 3:30pm**

Who to Contact

Craig Keithley
USB and FireWire Technology Evangelist
Keithley@apple.com

For More Information

- Core Audio Developer Services
- Mailing List—Core Audio API

  http://lists.apple.com/
- SDKs

  http://developer.apple.com/audio/

Q&A

Craig Keithley
USB & FireWire Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2002/urls.html

WWDC2002

WWDC2002

WWDC2002